UNIVERSIDADE ABERTA
www.uab.pt

Ministério da Educação e Ciência

U.C. 21046

Estruturas de Dados e Algoritmos Fundamentais

2 de julho de 2015

INSTRUÇÕES

- Leia estas instruções na totalidade antes de iniciar a resolução do teste.
- O enunciado do teste é constituído por 4 grupos de questões, tem **3** páginas e termina com a palavra FIM.
- O teste deve ser resolvido na sua totalidade em folhas de respostas, ficando o aluno com o enunciado.
- O teste é SEM CONSULTA. Todos os elementos necessários à resolução são fornecidos no enunciado.
- Utilize esferográfica azul ou preta para responder às questões. Respostas a lápis não serão consideradas.
- Nas respostas, tenha a preocupação de utilizar uma letra legível por outra pessoa.
- A interpretação dos enunciados das questões também faz parte da sua resolução, pelo que, se existir alguma ambiguidade, deve indicar claramente como foi resolvida.
- A correção do teste terá em conta critérios de proficiência e compreensibilidade do código ou pseudocódigo. Deve assinalar todas as opções tomadas. No código dos seus programas, todas as constantes, variáveis, métodos ou funções devem ser devidamente explicadas através de comentário.
- As respostas, que embora, sintáctica e semanticamente corretas, se apresentem pouco estruturadas serão severamente penalizadas, ou não consideradas. As respostas sem justificação serão fortemente penalizadas.
- Se o seu exemplar não estiver completo ou nele se verificar qualquer outra deficiência, por favor dirija-se ao professor vigilante.
- O não cumprimento das instruções implica a anulação das respetivas questões.
- O tempo de realização do teste é de 120 minutos, mais 30 minutos de tolerância.

1º Questão (5 Valores)

a) Construa uma função recursiva que dado um vetor com **N** elementos calcula a soma de todos os elementos desse vetor que são inferiores a um valor **K**. O cabeçalho da função é o seguinte: (2 val)

int somaInferioresK(int k, int n, int vec[])

```
int somaInferioresK(int k, int n, int vec[])
{
  if(n>0)
  {
    if(vec[n-1]<k)
      return vec[n-1]+ somaInferioresK(k, n-1, vec);
    else
      return somaInferioresK(k, n-1, vec);
  }
  else
    return 0;
}
```

b) Construa uma função iterativa que dado um vetor com **N** elementos indica o comprimento da maior subsequência crescente, em que todos os valores dessa sequência são inferiores a um valor **K**. Indique a complexidade do seu algoritmo. O cabeçalho da função é o seguinte: (2 val)

int maiorSubSeqCrescenteInfK(int k, int n, int vec[])

```
int maiorSubSeqCrescenteInfK(int k, int n, int vec[])
{
  int maior=0;
  int actual=0;
  for(int i=0;i<n-1;i++)
  {
     if(vec[i]<k)
     {
        if(vec[i+1]<k && vec[i+1]>vec[i])
            actual++;
        else
        {
            actual++;
            if(maior<actual)
               maior=actual;
            actual=0;
        }
     }
  }
 return maior;
}
```

A complexidade é O(n)

Ex: Suponha X=[1,4,0,5,7,2,8]; K=6; N=7
somaInferioresK(K,N,X) = 12
maiorSubSeqCrescenteInfK(K,N,X)=2

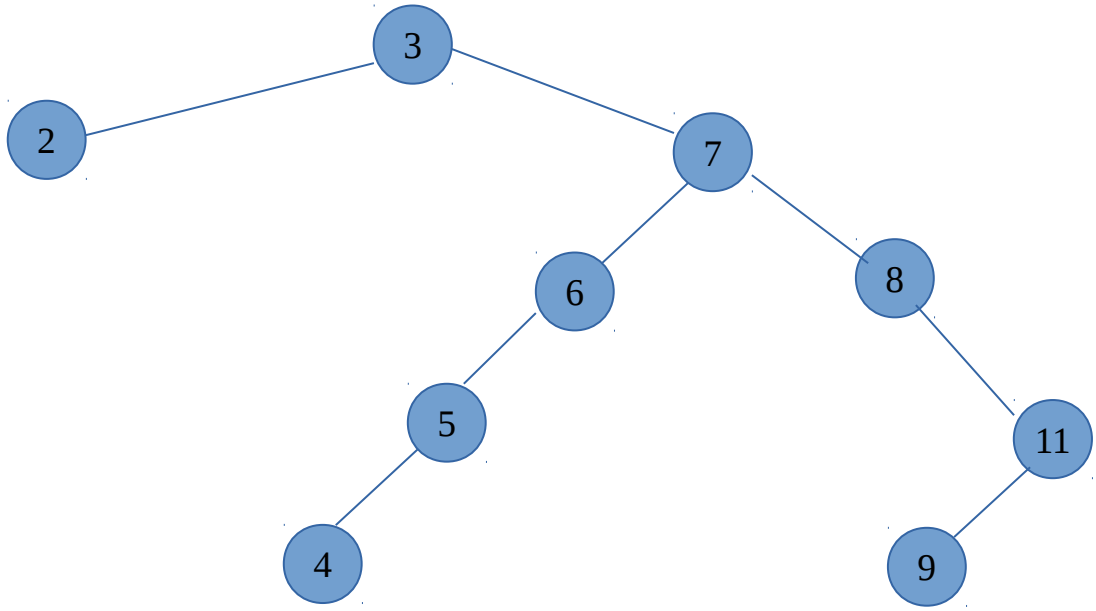
c) Indique a complexidade do seguinte algoritmo justificando. (1 val)

```
int soma=0;
for(int i=0; i<n; i++)
    for(int j=1; j<=i;j++)
        soma+=vec[i][j]
```

A complexidade é O(N^2)

2º Questão (5 Valores)

a) Considere uma árvore binária ordenada na qual qualquer nó da subárvore esquerda de **X** é menor que **X** e qualquer nó da subárvore direita de **X** é maior que **X**. Suponha que se insere a seguinte sequência de números: 3,7,8,2,6,5,11,9,4. Desenhe a árvore obtida. (2.5 val)



b) Qual a altura da árvore? (0.5 val)

A altura da árvore é 4 (distância máxima de qualquer folha à raiz)

c) Indique a ordem em que os elementos da árvore seriam visitados caso esta seja percorrida em pré-ordem (pre-order). (2 val)

O algoritmo para visita da árvore em pré-ordem é o seguinte:

```
void preOrdem(Struct No *pNo){
     if(pNo != NULL){
         visita(pNo);
         preOrdem(pNo→pEsquerda);
         preOrdem(pNo→pDireita);
     }
 }
```

Logo os elementos seriam visitados da seguinte forma:

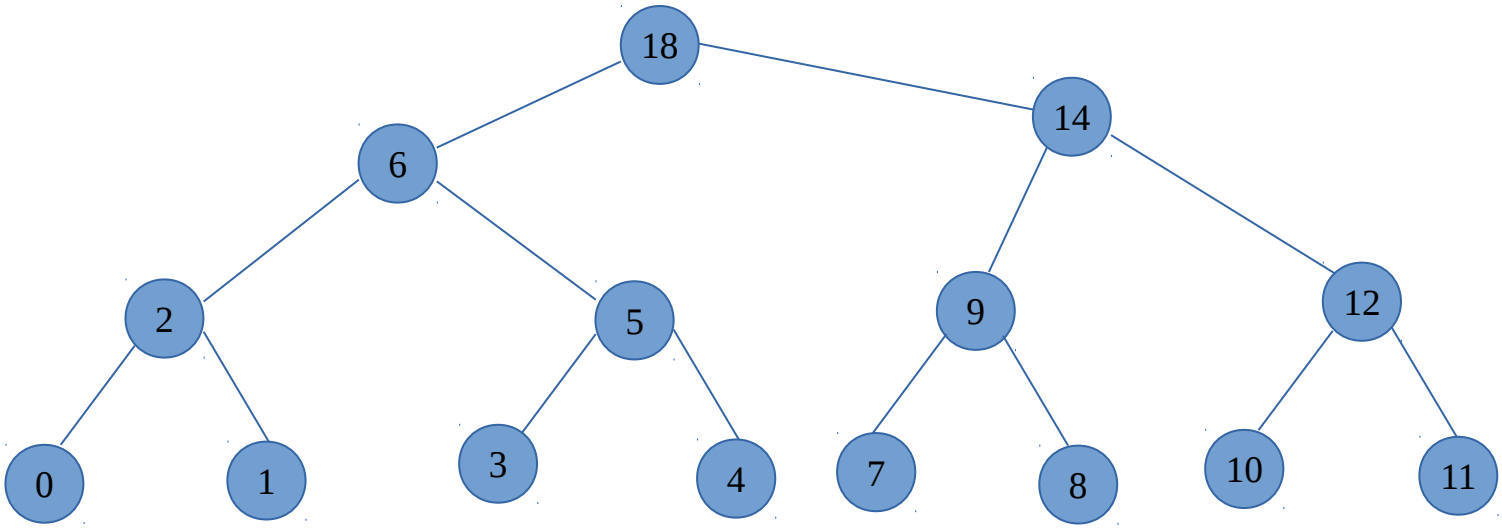
3, 2, 7, 6, 5, 4, 8, 11, 9

3º Questão (5 Valores)

Considere o seguinte Heap representado em vetor, no qual a prioridade é maior quanto maior o valor.

[18, 6, 14, 2, 5, 9, 12, 0, 1, 3, 4, 7, 8, 10, 11]

a) Desenhe a árvore binária correspondente a este Heap e explique porque motivo a condição da prioridade é mantida. (2 val)



A condição de prioridade é mantida pois cada nó ascendente é sempre maior que os nós descendentes (cada nó ascendente tem um valor mais alto do que os nós descendentes)

b) Implemente uma função verifica que dado um Heap representado por uma árvore retorna **true**, caso a condição de prioridade seja verificada e **false** no caso contrário. Suponha que estamos a verificar a prioridade num **max Heap.** (3 val)

```
struct No{
    int numero;
    struct No *esquerda;
    struct No *direita;
};
typedef struct No No;

bool verificaPrioridade(No* raiz)
{
    bool res=true;
    while (raiz!=NULL)
    {
     if(raiz->esquerda->numero > raiz->numero ||
        raiz->direita->numero > raiz->numero )
```

```
            return false;
        else
            return verificaPrioridade(raiz->esquerda) &&
                                        verificaPrioridade(raiz->direita);
    }
 return res;
}
```

4º Questão (5 Valores)

Considere uma tabela de dispersão **T** com comprimento 10 inicialmente vazia e as seguintes funções de dispersão.

h(x) = x % 10
g(x) = (x+1)%5

a) Considerando que as colisões são tratadas com hashing fechado, desenhe uma tabela de hash após a inserção das seguintes chaves (pela ordem apresentada): 25, 10, 35, 17 e 47. Justifique apresentando os cálculos efetuados. (3 val)

h(25)=25%10 = 5
h(10)=10%10 = 0
h(35)=35%10 = 5 (necessário fazer rehashing)
g(35)=(35+1)%5 = 1
h(17)= 17%10= 7
h(47) = 47%10=7 (necessário fazer rehashing)
g(47) = (47+1)%5 = 3

Aspeto final da tabela

10	35		47		25		17		

b) Indique a sequência de passos para a ordenação do seguinte vetor [3, 6, 2, 1, 8, 5] utilizando o algoritmo QuickSort. (2 val).

Passo 1: 3 6 2 1 8 5 Pivot = 3

Passo 2: 1 2 |3| 6 8 5 O pivot já está ordenado

Passo 3: 1 2 O sub-array esquerdo já está ordenado

Passo 4: 6 8 5 Pivot = 6

Passo 5: 5 |6| 8

Os sub-arrays direito e esquerdo estão ordenados

Passo 6: 1 2 3 5 6 8

FIM